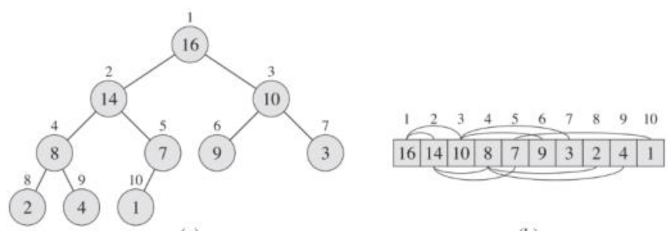
Final 复习

April 15, 2016 4:17 PM

	TOPIC	DETAIL
Week 1	Bound CLRS chapter 1 2 3 Chapter 3 growth of function	Worst case time complexity T(n) -> worst case time complexity 如果要证明worst case time complexity => T(n) is O(g(n)) IN SUMMARY: Let $T(n)$ be the *worst-case* time complexity of algorithm $\mathcal{A}$. 1. $T(n)$ is $O(g(n))$ iff $\exists c > 0, \exists n_0 > 0$, such that $\forall n \geq n_0$: for *every* input of size n, $\mathcal{A}$ takes *at most* $c \cdot g(n)$ steps. 2. $T(n)$ is $\Omega(g(n))$ iff $\exists c > 0, \exists n_0 > 0$, such that $\forall n \geq n_0$: for *some* input of size n, $\mathcal{A}$ takes *at least* $c \cdot g(n)$ steps. 3. $T(n)$ is $\Theta(g(n))$ iff $T(n)$ is $O(g(n))$ and $T(n)$ is $\Omega(g(n))$.

Week 2	CLRS chapter 6 heap
	Binomial heap
		CLRS chapter 6 heapsort
		
		6.1 heaps
		一些性质
		
		(a) (b)
		
		Heap 的index: left child的index是parent的两倍，right child的index是parent的两倍加1
		两种 => minheap, maxheap
		Maxheap property => A[parent(i)] >= A[i] ; root is the largest
		Minheap property 与之相反
		
		同一个heap的两个children之间不存在直接关系
		
		6.1 exercise
		
		6.1.1 min/ max num of elements in a heap of height h?
		Min: $2^{h-1}+1$
		Max: 2^h
		
		6.1.2 show that n element heap has height lgn(floor)
		Height equals logn when in a full heap => if the heap is nearly complete, it would be logn
		
		6.1.3 maxheap 性质导致root永远是最大的
		
		6.1.4 maxheap中最小的element必定在leaf中，因为parent > child, 所以如果它有child必定不是最小的element
		
		6.1.5 一个sorted order 的 array一定是min heap，但min heap并不要求array是sorted order的
		
		6.1.6 no 6有child 7
		
		6.1.7 leaf 起码占据所有node的一半（类似最下面一层
		
		6.2 maintain heap property
		
		MAX-HEAPIFY(A, i)
		l = LEFT(i)
		r = RIGHT(i)
		if $l \leq A.heap\text{-}size$ and $A[l] > A[i]$
		    $largest = l$
		else $largest = i$
		if $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$
		    $largest = r$
		if $largest \neq i$
		    exchange $A[i]$ with $A[largest]$
		    MAX-HEAPIFY$(A, largest)$
		Make sure the node originally in position I is in its right position after executing max-heapify

		Run time: O(logn) 6.3 Building a heap BUILD-MAX-HEAP Do max-heapify to all nodes except leaves, from bottom to top. O(nlogn) 6.4 the heapsort algorithm a. Build a max heap by build max heap b. exchange the root to the last position (we know root must be the largest) c. Do max heapify to the new root d. Recursively until there are two nodes left 6.5 priority queues Give each node a key, max heap Heap-maximum(A) Return the first node O(1) Heap-extract-max(A) Exchange 1 and last node Decrease the size Max heapify(A,1) O(logn) Heap-Increase-key(A,I,key) Change the key value of node in position I to the key in parameter and maintain the heap O(logn) MAX-HEAP-INSERT Binomial heap 具体结构 Min heap/ max heap 皆有可能 Find minimum => O(logn) Union => O(logn) Insert => O(logn) ExtractMax => O(logn) Extract Min
Week 3	BST AVL TREE CLRS: 12.1~12.3	BST Preorder traverse=> 中左右 Postorder traverse=> 左右中 In oder traverse=> 左中右 Search Find minimum/ maximum
Week 4	Hash table 13April Q1 Augmented data structure	Hash table Direct address table Search => Insert => Delete => These three operations all take O(1) Hash table Search => Insert => O(1) Delete => 1. Hash function => collide => chaining 2. SUHA 3. Load factor => n elements / m slots 4. Avg case time theta(1+alpha) Augmented data structure 1. Choose an underlying data structure and add additional information 2. Make sure the additional information can still be maintained in O(logn) times 3. Developing new operations In textbook => size = the number of nodes in the subtree; rank = node 从小到大排列的位置
Week 5	Probabilistic analysis Randomized algorithms 对于5.2还是不理解，知道结论，但对过程存在疑问，需要理解general solution. Probabilistic analysis 和 randomized algorithms 的区别 Related => HW4 Q1 part2	RQS (randomized quick sort) 取其中一个数作为pivot，分成比pivot大/小的两组，最坏情况需要runtime $O(n^2)$ Probabilistic analysis Tutorial => example P189 CLRS $E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr(i \text{ and } j \text{ are compared})$

	Quicksort Bloom filter CLRS chapter 5 , 7	

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Something close to $n \sum_{k=1}^{n} \frac{1}{k}$

$$\in \mathcal{O}(n \log n)$$

Analysis Over!

Randomized algorithms

Bloom filter
- False negative => 以为element 不在set中，实际上存在 （不会发生）
- False positive => 以为element 在set中，实际上不在（可能发生）=> 在设计过程中尽量减小这个可能性

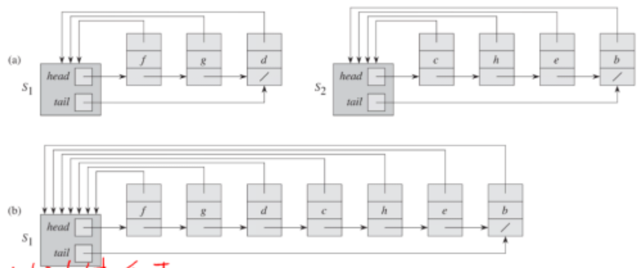
Week 6

Disjoint set

CLRS chapter 21

Collection of disjoint nonempty sets. Each set has distinguished elements. It has a representative.

Linked list representation
- 每一个set都有header,第一个node每一个node
- Makeset => theta(1)
- Findset => theta(1)
- Union => union connects two sets, 某一个set中的所有的 head会被替代



Lo.l-List 体系下

最差情况 => m sequence of operations on n objects take theta(n^2)

Two heuristic -> 改进措施

Weighted union
- Union的时候 把rank小的set安排在rank大的之下
- 更改union代码

Path compression
- 在findset的时候，把被find的东西直接放到find的结果下
- 更改find 代码

When applying both methods, the worst case running time is O(m alpha n) => alpha n grows very slow, <4

Week 7

Amortized analysis

CLRS Chapter 17

Amortized analysis => avg time required to perform a sequence of data structure

Aggregate method
- T(n)/n => avg cost of an operation in the worst case

Bit counter

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

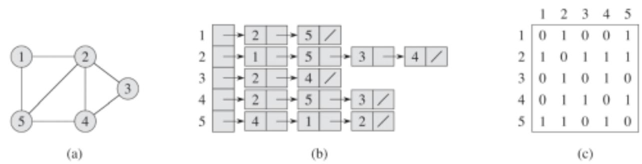
Accouting method
- 记录每一个行为需要的payment和credit

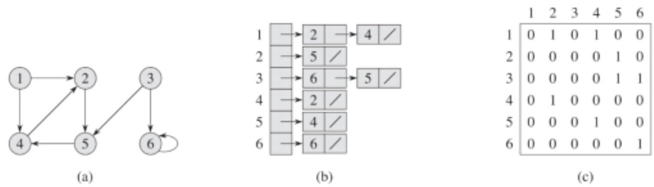
Week 8

CLRS 22.1 22.2

22.1 Representation of graphs
22.2 Breath first search

Representation of graphs
- Adjacency list
  - One list for each vertex, list to show the adjacent edge
- Adjacency matrix





Breath first search

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Running time O(V+E)
Initialization takes O(V)
Scanning adj list takes O(E)

Week 9

CLRS 22.3 22.4

Chapter 22.3 depth first search

Chapter 22.4 topological sort

22.3 depth first search

Running time of DFS => theta(V+E)

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1                // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]          // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                // blacken u; it is finished
time = time + 1
u.f = time
```

Theta(V + E)
DFS-visit => takes theta(V)
The other part => takes theta(E)

如果原图有back edge, 则表示图片是cyclic的

22.4 topological sort

Topological sort

根据finish time 从后往前

Week 10

Chapter 23 minimum spanning tree

Minimum spanning tree

Definition: the smallest tree that is acyclic and connects all of the vertices

Greedy approach

Find safe edge every time => cut the graph => cross the cut => find minimum weight light edge

Safe edge: theorem 23.1: light edge(weight is the minimum of any edge crossing the cut) crossing the cut

Kruskal algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

对所有的edges从小到大排序，并从小到大连接，如果已连接则跳过，直至所有的edges被连接

Sort all edges => O(ElogE)

For loops => O(E) find set and union operation + V make set operation => O((V+E)alpha(V))

Observing $E < v^2$ => lgE = O(lg V)

O(ElogV)

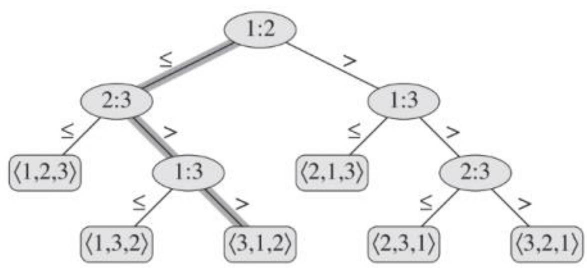
Prim algorithm

以某个点为起点，找最小的edge并连接，直到全部连接为止

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

O(ElogV) => can be improved to O(E + VlogV)

Week 11	CLRS chapter 35.2	Travelling salesman problem
Week 12		Decision tree/ problem complexity



Comparison based algorithm => at start of the of the algorithm, there are still n! possible permutation that could be correct sorted order. Each time we perform a comparison, we have only two output: true or false. Even in the best case, one split would eliminate half of the permutations. As a result, this perfect algorithm needs log(n!) comparisons = theta(nlogn)